ECOLE DES MINES DE SAINT ETIENNE
CAMPUS G. CHARPAK PROVENCE

Rapport

Conception Systèmes Numériques - Projet ASCON

Gevorg ISHKHANYAN

15 mai 2024

Table des matières

1 Introduction

Le chiffrement authentifié avec données associées (AEAD) est aujourd'hui largement employé dans les protocoles réseaux pour sécuriser les communications. Il assure à la fois la confidentialité et l'authentification des données échangées. Pour cela, le contenu du message est chiffré alors que l'en-tête reste en clair. Elle inclue des informations telles que les adresses IP source et destination, ainsi que la taille du message, permettant la redirection appropriée des paquets dans le réseau. L'algorithme ASCON128 garantit la confidentialité en chiffrant le contenu du message et assure également l'authenticité de l'en-tête grâce à un tag. Le destinataire doit vérifier ce tag ; en cas de non-correspondance avec la valeur attendue, le message est rejeté.

Une image qui représente bien ce chiffrement :

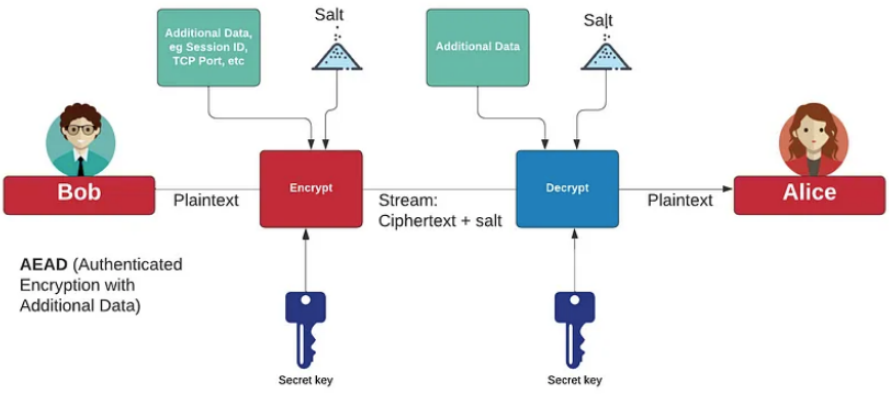

FIGURE 1 – Chiffrement AEAD

Au cours de ce projet, j'ai étudié en détail l'architecture de l'algorithme ASCON128, en explicitant les différentes étapes du processus de chiffrement, telles que l'initialisation de l'état, le traitement des données associées, le chiffrement du texte en clair et la génération du tag d'authentification. J'ai également examiné les spécifications des permutations, ainsi que les couches d'addition de constante, de substitution et de diffusion linéaire qui la composent. Malheureusement, je n'ai pas pu finaliser tout le projet. Néanmoins je montrerai mon travail effectué ainsi que ce que je prévoyais de faire pour la suite.

Ce rapport présentera en détail les différentes phases du projet, les choix de conception effectués, les résultats des simulations, ainsi que les éventuels points bloquants rencontrés et les solutions envisagées. L'objectif de ce projet était de comprendre le système ASCON128 avec notamment tous les modules qui le composent, de concevoir et de tester ces différents modules, mais également de découvrir et se familiariser avec un langage de description matérielle (Hardware Description Language) pour la première fois, qui est pour ce projet le langage SystemVerilog.

Pour réaliser ce projet, j'utiliserai le logiciel Modelsim pour compiler le code et simuler, ainsi que VSCode pour éditer mon code.

2 Le projet ASCON

2.1 Fonctionnement du système

Dans cette partie je vais expliquer le fonctionnement du système ASCON dans sa globalité, j'évoquerai les modules en détail dans la suite du rapport. Dans la suite du rapport, le terme "on XOR" ou "XORer" sera utilisé comme verbe pour indiquer l'utilisation de la fonction XOR (ou exclusif).
Cette version du système ASCON128 est une version volontairement simplifiée de l'agorithme original afin de permettre sa réalisation dans un temps limité. Toute la suite du rapport tient compte de cette simplification.

2.1.1 Les notations

On utilisera la terminologie et les notations suivantes :
- L'algorithme opère sur un état courant (ou state S) de 320 bits
- Cet état est mis à jour avec une opération appelée permutation. La permutation comprend soit 6 itérations, soit 12 itérations, et sera notée p6 (respectivement p12). Les itérations sont aussi appelées rondes.
- L'état se subdivise en deux parties :
  - Une partie dite externe de r = 64 bits, notée $S_r = S_{64}$
  - Une partie dite interne de c = 256 bits, notée $S_c = S_{256}$

L'état est donc la concaténation des deux parties, et S = $\{S_{64}, S_{256}\} = \{S_r, S_c\}$. L'état S de 320 bits est divisé en 5 tableaux registres x_i de 64 bits chacun, tel que représenté dans la Figure 2.
(Cette image provient du sujet du projet)

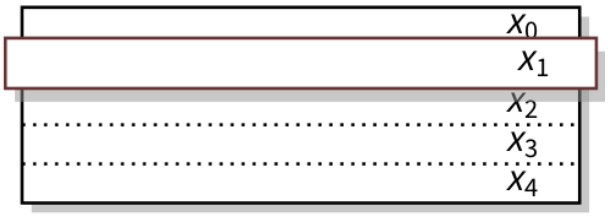

FIGURE 2 – Représentation de S = $\{x_0, x_1, x_2, x_3, x_4\}$

L'état peut aussi être interprété comme 64 colonnes de 5 bits chacune.
Notations supplémentaires :

- K : Clé secrète key K de 128 bits
- N : Nombre arbitraire nonce N de 128 bits
- T : Tag T de 128 bits
- P : Texte clair plaintext P de 184 bits
- C : texte chiffré ciphertext C de 184 bits
- A : Données associées associated data A de 48 bits
- IV : Vecteur d'initialisation initialisation vector IV de 64 bits 0x80400C0600000000

2.1.2 Les étapes de chiffrement

Le chiffrement se réalise en plusieurs étapes :
(Cette image provient du sujet du projet)

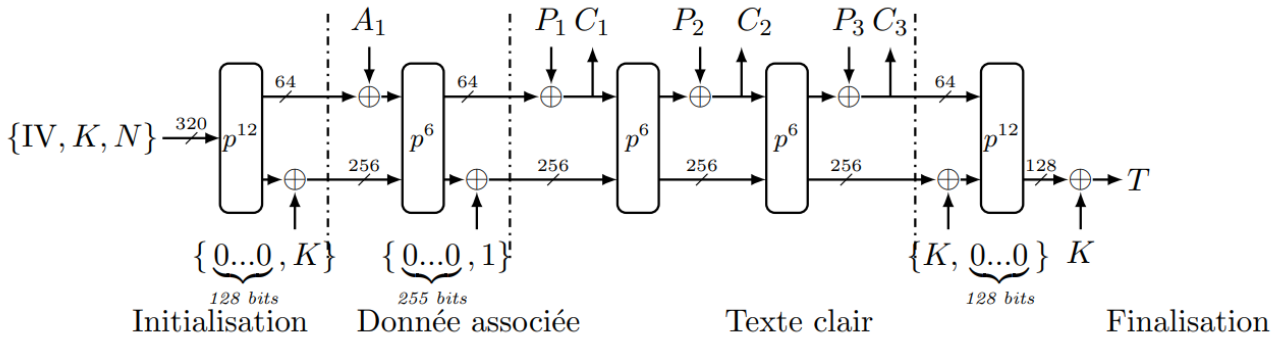

FIGURE 3 – Les étapes du chiffrement

La phase d'initialisation et d'association de données, sont les deux parties dans lesquelles nous allons préparer les données au chiffrement :

- Initialisation : On entre {IV, K, N} pour faire 12 permutations, puis on XOR (x3, x4) avec la clé K (xor_key)
- Donnée associée : On XOR les 64 premiers bits de la sortie d'initialisation avec la donnée associée A1, puis après avoir réalisé 6 permutations on XOR le bit de poids faible (lsb) de la sortie avec 1.

La partie Texte clair est la partie où on fait le chiffrement global du message, ce chiffrement du message est découpé en 3 petits chiffrements durant lesquelles on va XOR les parties du message (Plaintext P) avec les données associées, puis renvoyer le message chiffré (Cipher C). La phase de finalisation sert à communiquer le tag à l'autre partie pour le déchiffrement du message.

- Texte clair : On XOR les 64 premiers bits avec P1, puis on revoie directement ce résultat comme étant le texte chiffré.
- Finalisation : On XOR les 256 derniers bits (x1, x2, x3, x4) avec {K,0...0}, puis après avoir fait 12 permutations on XOR les 128 derniers bits (x3, x4) avec la clé K.

2.1.3 Le système global ASCON128

Le système ASCON128 (également appelé ASCON_Top) est composé de plusieurs blocs.

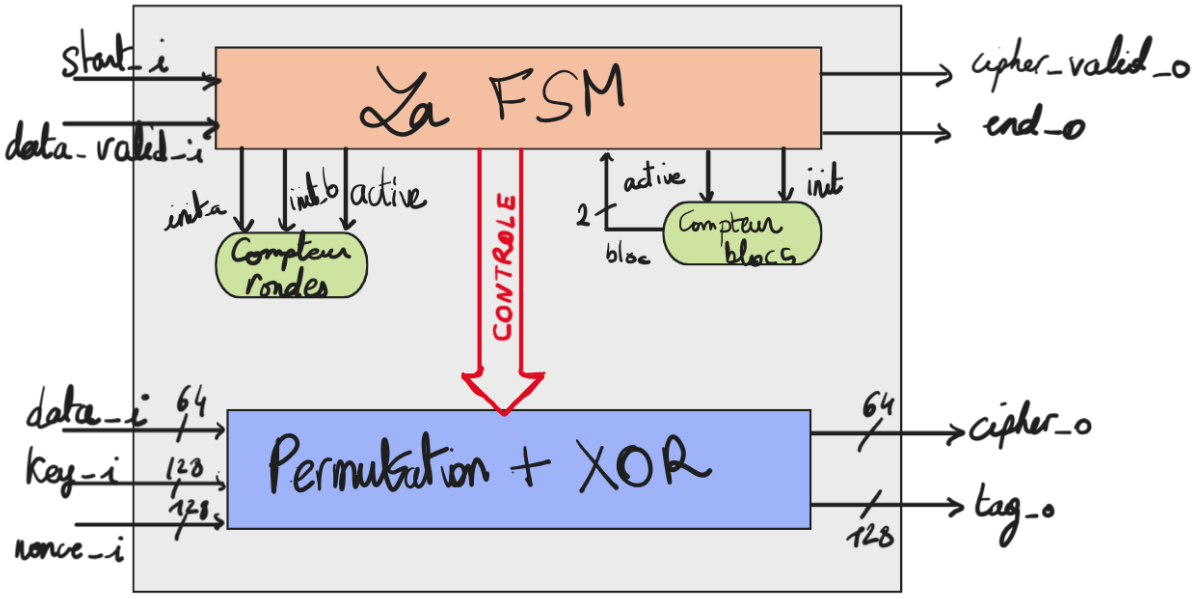

FIGURE 4 – Architecture du système ASCON128

Sans oublier les signaux resetb_i et clock_i comme entrée.

Nous pouvons faire le lien avec le cours d'architecture des processeurs. En effet car la FSM (Machine d'états finis) joue le rôle de "Control Path", tandis que la Permutation et les XOR jouent le rôle du "Data path". Cette transposition sur le cours m'a donné une autre vision qui m'a permis de mieux comprendre le fonctionnement du système. Je détaillerai leurs fonctionnements dans la suite.

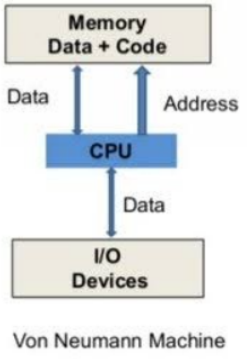

FIGURE 5 – Architecture Von Neuman

2.2 Organisation du travail

Afin d'avoir une meilleure organisation et ainsi bien se repérer dans le fichiers, les codes ont été classés dans des répertoires différents car il est important de séparer les fichiers compilés des codes sources et des testbench. Pour ce faire, je vais disposer les fichiers de la manière suivante :

- Le fichier compil_ascon.txt est celui utilisé pour compiler
- Les fichiers source sont dans SRC/RTL
- Les fichiers testbench sont dans SRC/BENCH
- Les fichiers exectuables et les librairies compilées sont dans LIB/LIB_RTL et dans LIB/LIB_BENCH
- Le reste des fichiers en dessous de compil_ascon.txt ne seront pas utilisés, car j'utilise directement ModelSim en local, et non depuis un serveur externe (comme *tallinn* par exemple).

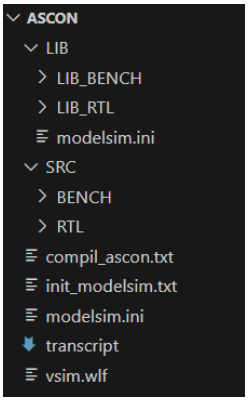

FIGURE 6 – Arborescence des fichiers code

2.3 La compilation et simulation

La compilation se fait par le script de compilation qui se trouve dans compil_ascon.txt. Au lancement de Modelsim on se trouve dans le dossier suivant :

```
ModelSim> pwd
# C:/intelFPGA/20.1
```

FIGURE 7 – Dossier de départ ModelSim

Il faut alors enlever les commentaires des testbench que l'on souhaite compiler et simuler, puis de lancer les commandes suivantes dans le terminal de ModelSim :

```
# Aller dans le dossier ASCON, pour moi sur Windows ce sera :
cd ../../Users/"utilisateur"/Desktop/ASCON

# Puis lancer le script de compilation
source compil_ascon.txt
```

Puis observer les résultats de la simulation.

3 Les éléments du projet

Les composantes principales de l'algorithme ASCON128 sont les deux permutations p^6 et p^{12} de l'état S sur 320 bits. Ces deux permutations appliquent itérativement la transformation de ronde p, qui se compose des trois étapes P_C, P_S, et P_L tel que :
$P = P_C \circ P_S \circ P_L$. Les permutations p^6 et p^{12} différent uniquement par le nombre de rondes.

Voici un schéma de la permutation en entier. Cette version embarque également les XOR de début et de fin, ainsi que les registres pour les ciphers et le tag :

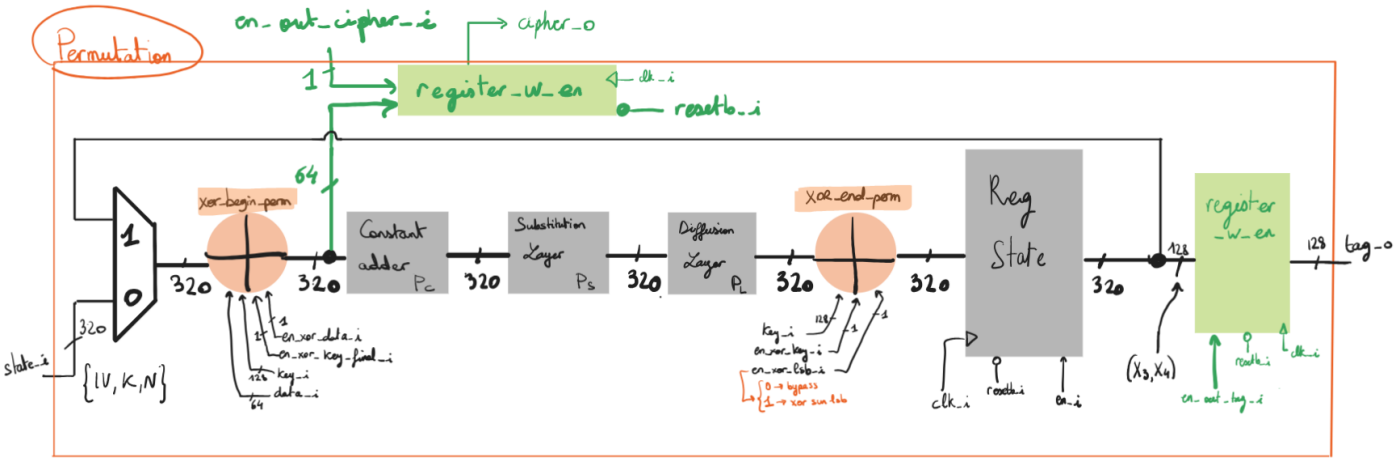

FIGURE 8 – Une permutation complète

3.1 Addition de la constante

L'addition de constante P_C ajoute une constante de ronde c_r au registre x_2 de l'état S au cours de la ronde i, tel que $x_2 \longleftarrow x_2 \bigoplus c_r$. Les constantes sont définies en annexe, sur la Figure 16. Il faut pour cela ajouter les constantes dans le package ascon_pack

```
1     localparam logic [7:0]   round_constant [0:11] = {8'hF0, 8'hE1, 8'
         hD2, 8'hC3, 8'hB4, 8'hA5, 8'h96, 8'h87, 8'h78, 8'h69, 8'h5A, 8'
         h4B};
```

Listing 1 – Ajout des constantes dans ascon_pack.sv

On remarque que le *state_o* est bien modifié comme attendu par rapport au *state_i*.

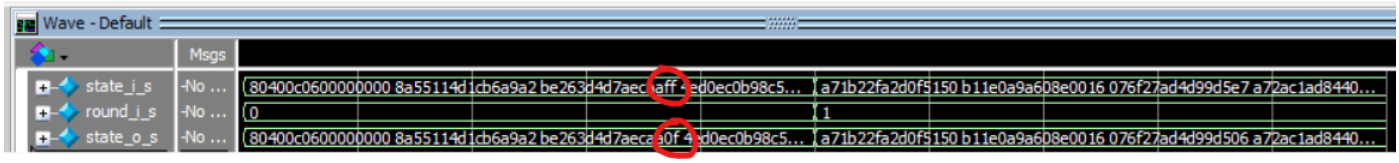

FIGURE 9 – Chronogramme de constant_addition_tb.sv

3.2 Couche de substitution

La couche de substitution P_S modifie l'état S en appliquant en parallèle la substitution de 5 bits en colonne en utilisant la table de substitution S-box définie en Annexe sur la Figure 17. L'état S s'interprète dans ce cas-là comme un tableau de 64 colonnes, comme indiqué dans la partie notations. Il faut pour cela ajouter le tableau S-box dans le package ascon_pack. Il faut d'abord faire le module ascon_sbox.sv pour appliquer la fonction $S(x)$ puis faire le module substitution_layer.sv

```
1     localparam logic [4:0]   sbox_t [0:31] = {5'h04, 5'h0B, 5'h1F, 5'h14
          , 5'h1A, 5'h15, 5'h09, 5'h02, 5'h1B, 5'h05, 5'h08, 5'h12, 5'h1D,
          5'h03, 5'h06, 5'h1C, 5'h1E, 5'h13, 5'h07, 5'h0E, 5'h00, 5'h0D, 5'
          h11, 5'h18, 5'h10, 5'h0C, 5'h01, 5'h19, 5'h16, 5'h0A, 5'h0F, 5'
          h17 };
```

Listing 2 – Ajout du tableau S-box dans ascon_pack.sv

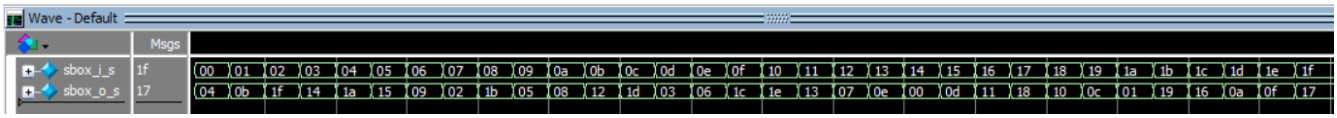

FIGURE 10 – Chronogramme de ascon_sbox_tb.sv

On remarque que le *sbox_o_s* est bien modifié comme attendu par rapport au *sbox_i_s*, le résultat est identique au tableau de la S-box.

Naturellement, la couche de substitution fonctionne bien d'après le testbench étant donnée que la S-box est bien fonctionnel.

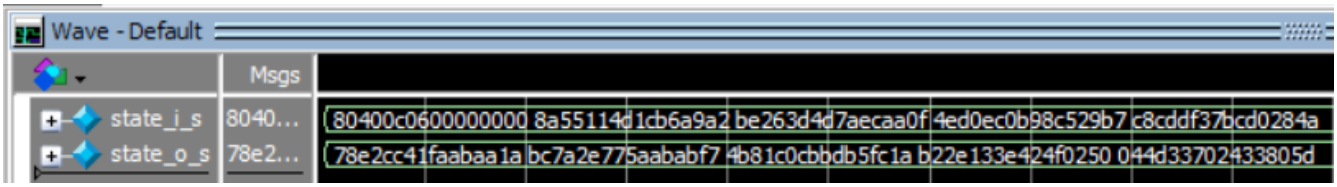

FIGURE 11 – Chronogramme de substitution_layer_tb.sv

3.3 Couche de diffusion

(Cette image provient du sujet du projet)

La couche de diffusion linéaire p_L applique une diffusion à chacune des lignes de 64 bits ou registres x_i. On applique ainsi l'opération suivante :

$$x_i \leftarrow \Sigma_i(x_i)$$

$$\begin{array}{lllllllllllll}
x_0 & \leftarrow & \Sigma_0(x_0) & = & x_0 & \oplus & (x_0 & \ggg & 19) & \oplus & (x_0 & \ggg & 28) \\
x_1 & \leftarrow & \Sigma_1(x_1) & = & x_1 & \oplus & (x_1 & \ggg & 61) & \oplus & (x_1 & \ggg & 39) \\
x_2 & \leftarrow & \Sigma_2(x_2) & = & x_2 & \oplus & (x_2 & \ggg & 1) & \oplus & (x_2 & \ggg & 6) \\
x_3 & \leftarrow & \Sigma_3(x_3) & = & x_3 & \oplus & (x_3 & \ggg & 10) & \oplus & (x_3 & \ggg & 17) \\
x_4 & \leftarrow & \Sigma_4(x_4) & = & x_4 & \oplus & (x_4 & \ggg & 7) & \oplus & (x_4 & \ggg & 41)
\end{array}$$

(Cette image provient du sujet du projet)

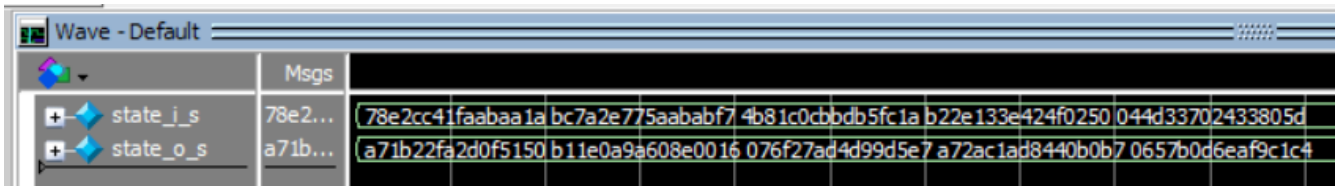

FIGURE 12 – Chronogramme de diffusion_layer_tb.sv

On remarque que le *state_o_s* est bien modifié comme attendu par rapport au *state_i_s*, le résultat est identique au résultat attendu.

3.4 La permutation

J'ai d'abord réalisé une permutation_step_1.sv afin de comprendre le fonctionnement de la permutation. Ensuite j'ai ajouté les XOR du début et de fin, ainsi que les registres pour les ciphers et le tag. J'expliquerai ces ajouts par la suite.

Les valeurs attendues pour la permutation (ici pour la permutation 11 par exemple) sont celles ci :

```
1    9de1e2d04b50bd86 13141c888a702ce4 55aa22c50d252d83 6530ee4949ecd5b1
       1209bee19f4a7ade
```

Listing 3 – Valeur attendue de state_o a la 11eme permutation

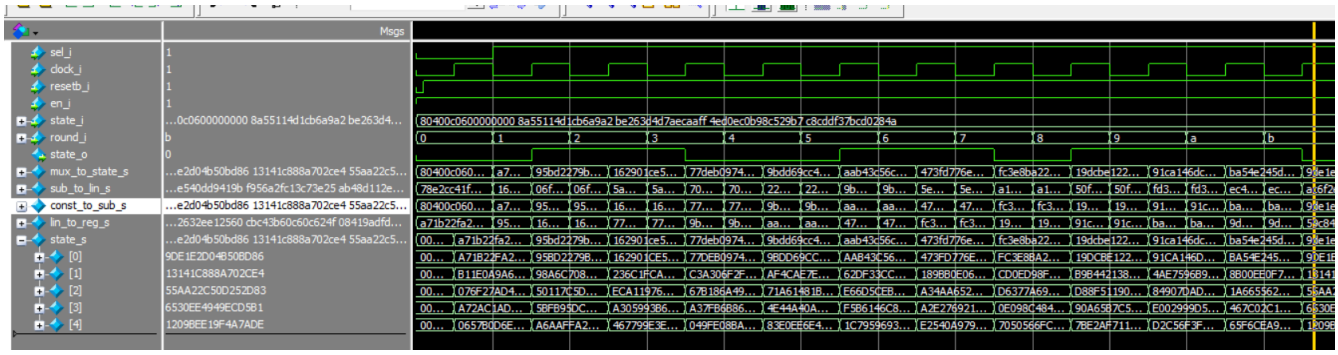

FIGURE 13 – Chronogramme de permutation_step_1_tb.sv

Nous avons bien exactement les mêmes valeurs que celles attendues, donc la permutation seule fonctionne bien.

3.5 La permutation avec XOR et registres ajoutés

Le bloc XOR begin est utilisé durant les phases de Données associées, de Texte clair et de Finalisation. Ce module a pour rôle d'effectuer une opération XOR entre le signal d'entrée en sortie du multiplexeur et la clé K ou la Donnée associée. Dans ce cas, si en_xor_data_i est activé l'opération se fait avec la donnée et si en_xor_key_i l'opération est effectuée avec K.

Par ailleurs, les phases d'Initialisation, de Données Associées et de Finalisation se servent du bloc XOR end. Ce module fonctionne de la même manière que le XOR begin à la différence près qu'il effectue l'opération entre le signal en sortie du multiplexeur et la clef ou le dernier bit de x_4 avec un bit 1. Les signaux d'activation sont respectivement en_xor_end_key_i et en_xor_lsb_i.

Afin de bien réaliser la permutation avec les XOR et les registres, il faut veiller à faire des déclarations internes, c'est à dire des "fils" qui vont permettre de connecter nos modules entre eux, en utilisant le même fil pour la sortie d'un module et l'entrée d'un autre.

```
1       // Internal net declaration
2 type_state mux_to_xor_begin_s, xor_begin_to_add_s,
3  const_to_sub_s, sub_to_lin_s, lin_to_xor_end_s, xor_end_to_reg_s;
```

Listing 4 – Pour faire les connexions internes

Après avoir ajouté les XOR et les registres, on refait la simulation pour une permutation.

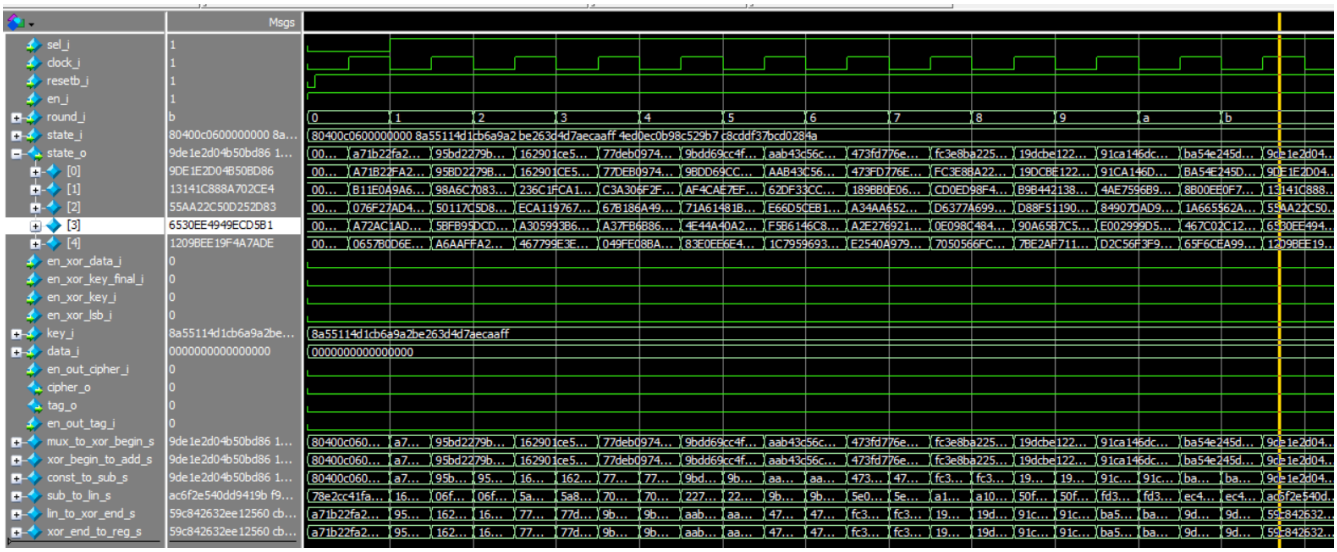

FIGURE 14 – Chronogramme de permutation_tb.sv

Nous avons bien exactement les mêmes valeurs que celles attendues, donc la permutation complète fonctionne.

3.6 La FSM

Je n'ai pas pu finaliser la FSM pour tout le chiffrement (Initialisation, Donnée associée, Texte clair, et Finalisation), néanmoins j'ai réalisé la partie d'initialisation du chiffrement. Cette FSM est dans notre cas une machine de Moore, car les états futurs dépendent unquement des états précédents.

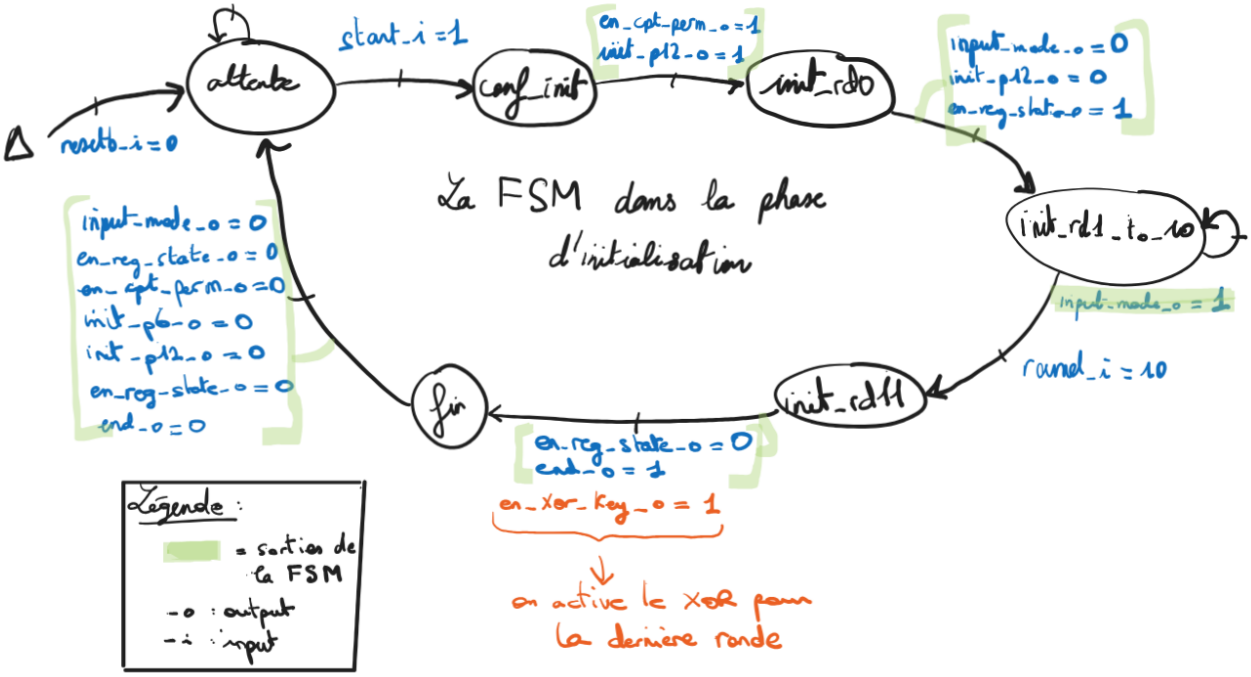

FIGURE 15 – Initialisation de la FSM

4 Conclusion

Pour conclure, ce projet de Conception de Systèmes Numériques sur ASCON128 à été une expérience très stimulante. J'ai réussi à implémenter la permutation complète avec son testbench afin de vérifier son bon fonctionnement.

Le projet n'ayant pas complétement abouti, la suite serait naturellement de finaliser la FSM globale (et non uniquement pour la phase d'initialisation), ainsi que la partie ASCON_TOP pour lier tous ces modules entre eux. Néanmoins le plus dur à été réalisé avec la permutation et les registres et XOR implémentés dedans. En plus de finaliser la FSM, il serait intéressant d'essayer d'optimiser les performances en optant pour une machine de Mealy à la place d'une machine de Moore (encore une fois pour faire le lien avec le cours d'architecture des processeurs), mais à voir biensur si cette solution est compatible avec nos exigences en terme de fiabilité notamment au niveau électrique.

Pour finir, ce projet m'a permis d'acquérir des connaissances et des compétences non négligeables dans le domaine de la conception de systèmes numériques. Je suis très heureux d'avoir pu faire ce projet en première année du cursus ISMIN, car ce changement

va me permettre d'avoir déjà des compétences utiles pour un potentiel stage de deuxième année. C'est également une première porte d'entrée dans la cryptographie, moi qui n'a jamais expérimenté ça auparavant.

Je tiens à exprimer ma gratitude envers toutes les personnes qui m'ont appris beaucoup de choses sur ce sujet, notamment mon professeur **Jean-Max DUTERTRE**. Mais également mon camarade **Ibrahim HADJ-ARAB** qui m'a quelques fois aidé pour certains debugs.

5 Ressources / Annexe

Figure 1 : https ://medium.com/asecuritysite-when-bob-met-alice/asconrust-and-aead-b237afb78a83

Figure 4 : https ://www.quora.com/Which-is-the-most-commonly-usedcomputer-architecture-von-Nuemann-or-Harvard-architecture-and-why

Tableau des constantes pour l'addition de la constante :

Ronde r de p^{12}	Ronde r de p^6	Constante c_r
0		00000000000000000f0
1		00000000000000000e1
2		00000000000000000d2
3		00000000000000000c3
4		00000000000000000b4
5		00000000000000000a5
6	0	0000000000000000096
7	1	0000000000000000087
8	2	0000000000000000078
9	3	0000000000000000069
10	4	000000000000000005a
11	5	000000000000000004b

FIGURE 16 – Addition de constante, (Cette image provient du sujet du projet)

x	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
$S(x)$	04	0B	1F	14	1A	15	09	02	1B	05	08	12	1D	03	06	1C	1E	13	07	0E	00	0D	11	18	10	0C	01	19	16	0A	0F	17

FIGURE 17 – Tableau de substitution S_box, (Cette image provient du sujet du projet)